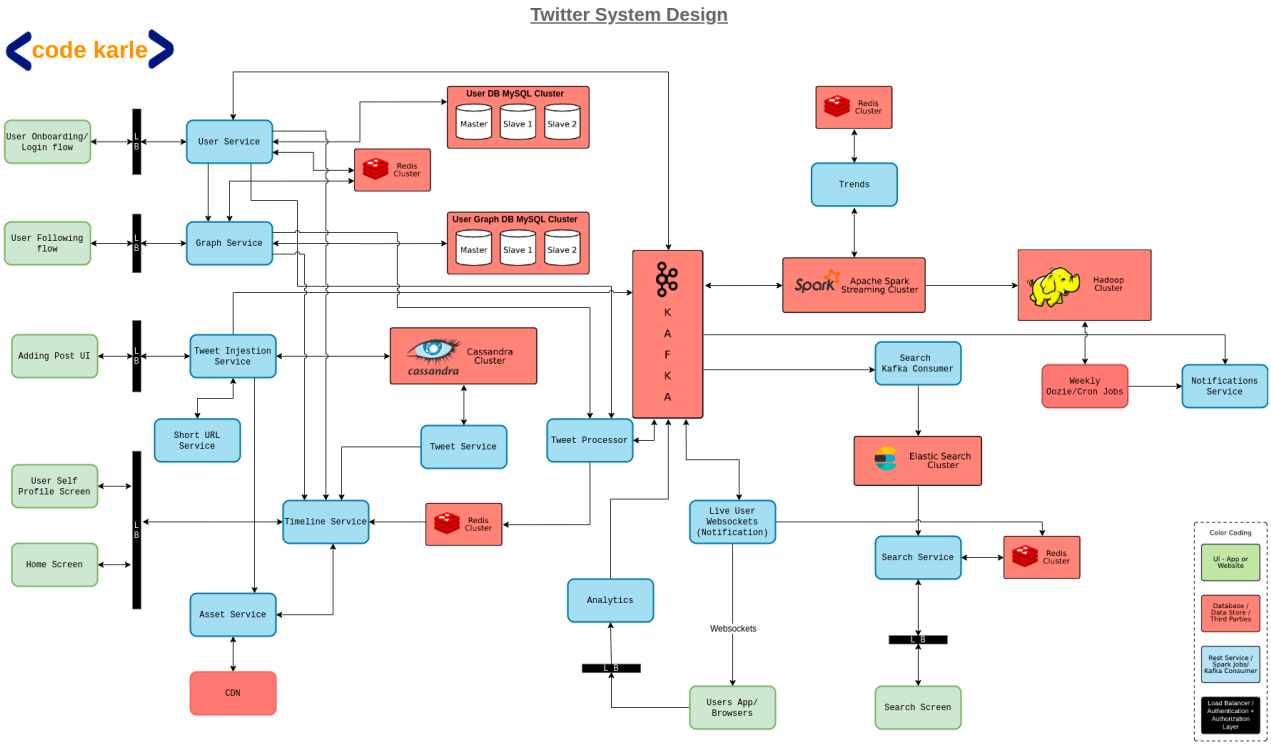
X

X is a social media platform that allows users to send and receive short messages called “tweets.” The platform has millions of users around the world, and must be able to handle a high volume of tweets and other data in real time.

Architecture

At the core of X’s architecture is a distributed database that stores all of the tweets and other data in the system. This database is optimized for fast read and write operations, as well as scalability to support a large number of users.



Microservices involve breaking down the application into smaller, independent services that can be developed, deployed, and scaled independently. This approach allows for greater flexibility, scalability, and maintainability.

Programming languages

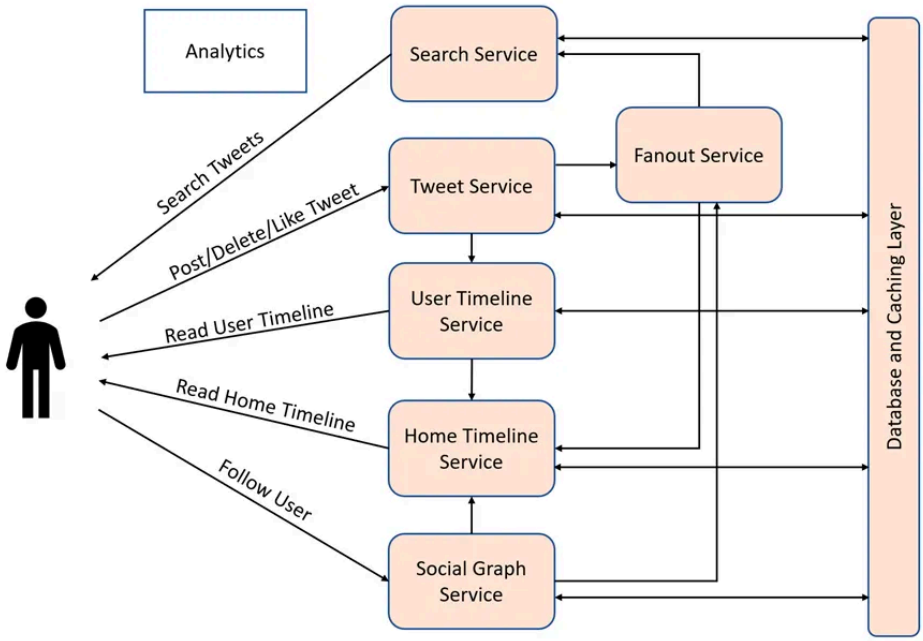
Some of the key programming languages used by X include:

- **Java:** Twitter primarily uses Java for back-end systems, large-scale and distributed systems, particularly for its core systems such as the tweet storage and retrieval system.
- **Scala:** Twitter has also heavily invested in the Scala programming language. It is used for building many of its back-end services and tools, particularly for its real-time data processing and machine learning pipelines.
- **JavaScript:** JavaScript is being used for building the front-end of web and mobile applications.
- **Ruby:** Ruby is used for building many of Twitter's internal tools and utilities, as well as for its developer API.

Tweet Service

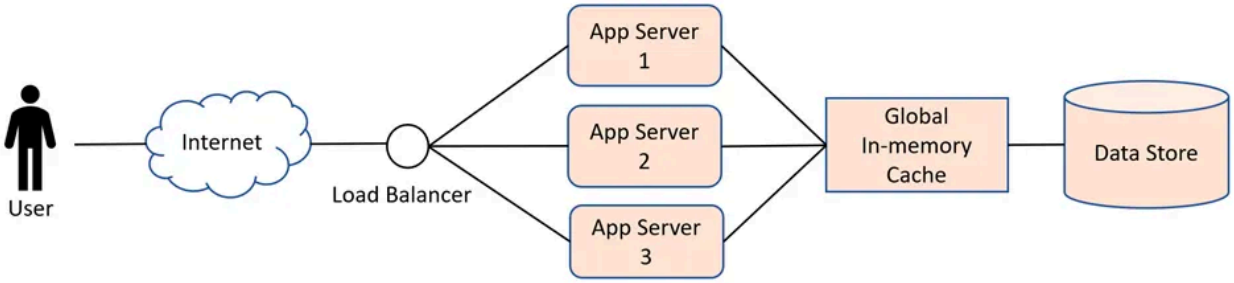
The X service comprises several micro-services as follows:

- Tweet Service
- User Timeline Service
- Fanout Service
- Home Timeline Service
- Social Graph Service
- Search Service



Tweet Service

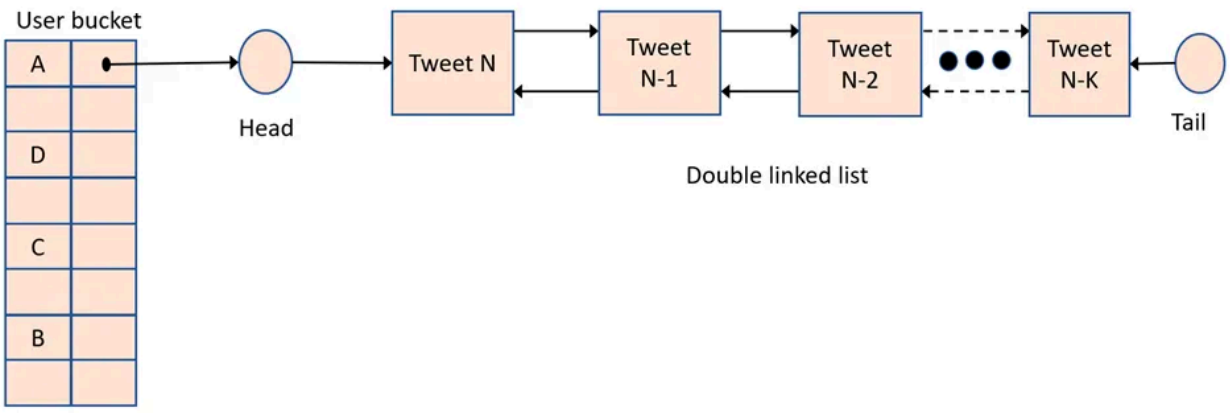
The Tweet Service is responsible for receiving user tweets and forwarding them to the timeline and search services. It also stores the users and tweet information, including the number of tweets from a user, user likes, etc.



User Timeline Service

The User timeline service is responsible for returning the user his timeline, which contains all the user tweets in descending order of time. This service is used whenever a user either looks at his timeline or some other user's timeline.

This service comprises the application servers and the distributed in-memory cache. There is no datastore involved in this service as we really do not need one.



Fanout Service

The Fanout service comprises multiple distributed queues.
Whenever a user sends a tweet message, the Fanout service enters the message in the queue for the tweets. Later, it queries the social graph service to get the list of followers of the user (sender of the tweet) and then insert as many messages in the second set of queues (i.e., queues for each tweet to followers) as the number of followers.

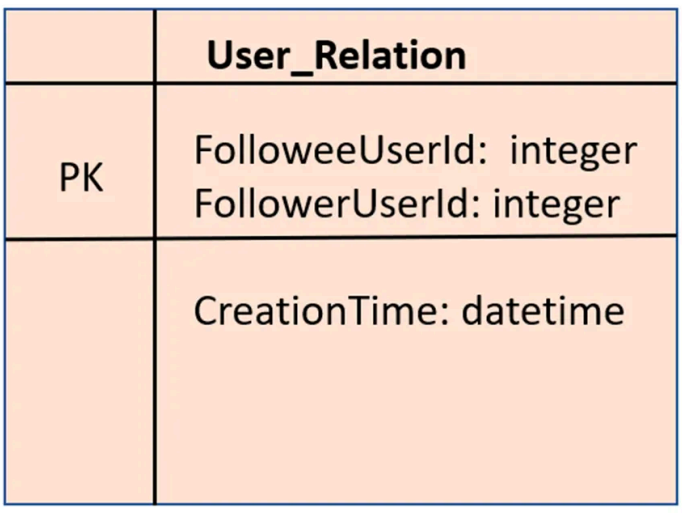
Home Timeline Service

The Home Timeline service is responsible for displaying the user's home timeline. This comprises all the tweets from the other users, which the current user is following. The tweets are displayed in descending order of time.

The home timeline service is a bit more complex than the user timeline service, though. The user timeline service always inserts new tweets at the head and removes the oldest tweet when the list reaches its maximum size. However, in the home timeline, since the user is following many other users (called followee), the service needs some mechanism to rank the tweets from different followee users.

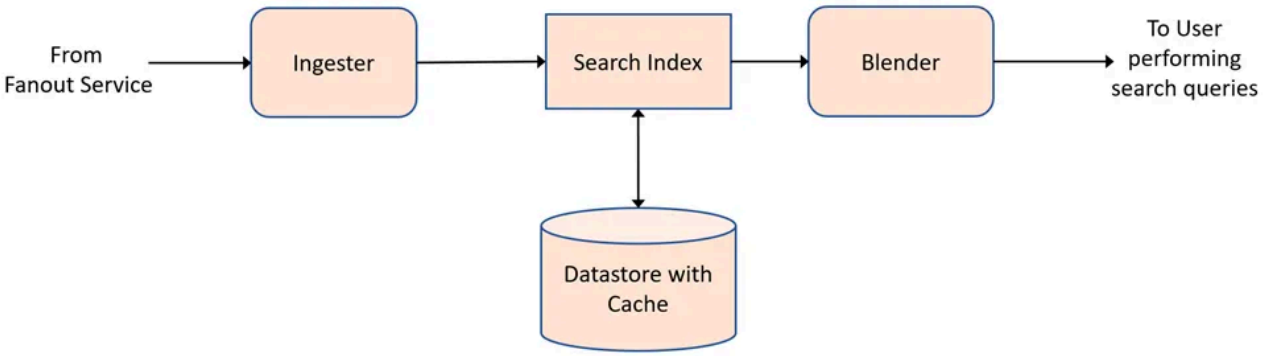
Social Graph Servce

The social graph service keeps track of which users are following which users. This service implements the following APIs.
The Social Graph Service will have almost the same design as the Tweet service with application servers, a distributed cache, and a datastore. It is also using the distributed cache in a write-through manner.
The following is the database schema that is used to store user relationships.



Search Service

This service is responsible for serving users' search queries. The Fanout service passes the tweet to the search service. The search service has the following logical components (or micro-services).



Ingester
The Ingester (or Ingestion Engine) is responsible for tokenizing the tweets into a number of tokens or terms (or keywords).

Search Index
The Search Index micro-service will create an inverted index. An inverted index is an index data structure storing a mapping from the content, such as words, to its location in a document or set of documents, which in our case, is a tweet or set of tweets. For this, the search index micro-service store the words and the tweet Id in a table in the datastore.

Blender
Blender serves the search queries by the users on the Twitter platform. When a search query comes, it first determines the search terms. It then also does a process of stemming to determine the root words of each search term and then uses the root words to run search queries on the inverted index of terms.

X Databases

X started with MySQL as the primary data store, from a single instance the persistence layer grew to a large number of clusters.

X has had one of the biggest deployments of MySQL right from its inception. It has MySQL clusters with thousands of nodes serving millions of queries per second.

MySQL has primarily two use cases:

1. Acting as the storage node for the distributed data store within Twitter's sharding framework. MySQL storage nodes provide reliability & performance in the overall distributed store.

2. Powers services such as ads, authentication, Twitter trends & many internal services.

The engineering team at Twitter has also built a scalable MySQL database service framework called Mysos.

It is based on Apache Mesos. Mesos enables Mysos to schedule, monitor & communicate with MySQL instances in the cluster. MySQL instances were sharded with the help of a framework for creating distributed datastores called Gizzard.